Слайс-жесты.

- Движения типа слайс и элемент
    - Push / Добавить элемент x в конец слайса
    - Pop / Выдернуть последний элемент слайса
    - Push Front/Unshift / Вставить элемент x в начало
    - Pop Front/Shift / Выдернуть первый элемент слайса
    - Insert / Вставить x на i-ю позицию
    - Insert / Вставить x на i-ю позицию.
    - Get / Выдернуть i-й элемент слайса
    - Delete / Удалить i-й элемент, сохраняя порядок следования оставшихся
    - Safe delete/ Удалить i-й элемент, сохраняя порядок следования оставшихся - безопасная (корректная) версия.
    - Safe delete last/ Безопасно удалить последний элемент.
    - Safe delete front/ Безопасно удалить начальный элемент.
    - Delete without preserving order / Удалить i-й элемент, не сохраняя порядок следования оставшихся
    - Safe delete without preserving order / Корректно удалить i-й элемент, не сохраняя порядок следования оставшихся
- Движения типа слайс и слайс
    - Append slice / Прицепить b в конец a
    - Copy / Копировать a в b
    - Cut / Удалить отрезок a[i:j]
    - Safe cut / Безопасно удалить отрезок a[i:j]
    - Extend / Добавить в конец слайса пустой отрезок длины j
    - Expand / Вставить пустой отрезок длины j внутрь слайса, начиная с позиции i
    InsertVector / Вставить слайс b внутрь слайса a, начиная с позиции i
- Другие операции
    - Reversing / Переворачивание слайса задом наперёд "на месте" - без перераспределения памяти.
    - Filtering without allocating / Фильтрация "на месте"
    - Batching with minimal allocation / Расщепление слайса на пакеты данных фиксированной длины
    - Move to front, or append if not present, in place / Поиск первого вхождения заданного элемента и перемещение его вперёд (без перераспределения памяти). Если такого элемента не было, то добавляем его в конец слайса.

Движения типа слайс и элемент

Push / Добавить элемент x в конец слайса

```
a = append(a, x)
```

Pop / Выдернуть последний элемент слайса

```
x, a = a[len(a)-1], a[:len(a)-1]
```

Push Front/Unshift / Вставить элемент x в начало

```
a = append([]T{x}, a...)
```

Pop Front/Shift / Выдернуть первый элемент слайса

```
x, a = a[0], a[1:]
```

Insert / Вставить x на i-ю позицию

Менее эффективно:

```
a = append(a[:i], append([]T{x}, a[i:]...)...)
```

Более эффективно:

```
a = append(a, 0 /*нулевое значение типа элементов слайса*/)
copy(a[i+1:], a[i:])
a[i] = x
```

Что-то промежуточное:

```
a = append(a[:i+1],a[i:]...)
a[i] = x
```

Get / Выдернуть i-й элемент слайса

```
x, a = a[i], append(a[:i], a[i+1:]...)
```

либо

```
x, a = a[i], a[:i+copy(a[i:], a[i+1:])]
```

Delete / Удалить i-й элемент, сохраняя порядок следования оставшихся

```
a = append(a[:i], a[i+1:]...)
```

либо

```
a = a[:i+copy(a[i:], a[i+1:])]
```

Safe delete/ Удалить i-й элемент, сохраняя порядок следования оставшихся - безопасная (корректная) версия.

```
copy(a[i:], a[i+1:])
a[len(a)-1] = nil // нулевое значение типа элементов слайса
a = a[:len(a)-1]
```

Safe delete last/ Безопасно удалить последний элемент.

a[len(a)-1] = nil // нулевое значение типа элементов слайса
a = a[:len(a)-1]

Safe delete front/ Безопасно удалить начальный элемент.

a[0] = nil // нулевое значение типа элементов слайса
a = a[1:]

Delete without preserving order / Удалить i-й элемент, не сохраняя порядок следования оставшихся

```
a[i] = a[len(a)-1]
a = a[:len(a)-1]
```

Safe delete without preserving order / Корректно удалить i-й элемент, не сохраняя порядок следования оставшихся

```
a[i] = a[len(a)-1]
a[len(a)-1] = nil //или нулевое значение типа элементов слайса
a = a[:len(a)-1]
```

Движения типа слайс и слайс

Append slice / Прицепить b в конец a

```
a = append(a, b...)
```

Copy / Копировать a в b

```
b = make([]T, len(a))
copy(b, a)
```

Cut / Удалить отрезок a[i:j]

```
if i <= j  { a = append(a[:i], a[j:]...) }
```

Safe cut / Безопасно удалить отрезок a[i:j]. Условие i<=j не проверяется.

```
copy(a[i:], a[j:])
for k, n := len(a)-j+i, len(a); k < n; k++ {
    a[k] = nil // или нулевое значение типа элементов слайса
}
a = a[:len(a)-j+i]
```

Extend / Добавить в конец слайса пустой отрезок длины j

```
a = append(a, make([]T, j)...)
```

Expand / Вставить пустой отрезок длины j внутрь слайса, начиная с

позиции i

```
a = append(a[:i], append(make([]T, j), a[i:]...)...)
```

InsertVector / Вставить слайс b внутрь слайса a, начиная с позиции i

a = append(a[:i], append(b, a[i:]...)...)

Более эффективно

```
func Insert(s []T, k int, vs []T) []T {
    if n := len(s) + len(vs); n <= cap(s) {
        s2 := s[:n]
        copy(s2[k+len(vs):], s[k:])
        copy(s2[k:], vs)
        return s2
    }
    s2 := make([]int, len(s) + len(vs))
    copy(s2, s[:k])
    copy(s2[k:], vs)
    copy(s2[k+len(vs):], s[k:])
    return s2
}
a = Insert(a, i, b)
```

Другие операции

Reversing / Переворачивание слайса задом наперёд

Операция выполняется “на месте” - без перераспределения памяти.

```
// Variant 1.
last:= len(a) - 1
for i := len(a)/2-1; i >= 0; i-- {
    a[i], a[last-1] = a[last-i], a[i]
}
```

```
// Variant 2.
```

```
for left, right := 0, len(a)-1; left < right; left, right = left+1, right-1 {
    a[left], a[right] = a[right], a[left]
}
```

Filtering without allocating / Фильтрация “на месте”

Фильтр - булевская функция keep: `func keep(x SliceElementType) bool`

```
// Variant 1.
n := 0
for _, x := range a {
    if keep(x) {
        a[n] = x
        n++
    }
}
// if there exist elements which must be garbage
// collected, the following cycle can be included
for i:= n; i<len(a); i++  {
    a[i] = nil // or the zero value of T
}
a = a[:n]
```

```
// Variant 2.
b := a[:0]
for _, x := range a {
    if f(x) {
        b = append(b, x)
    }
}
// if there exist elements which must be garbage
// collected, the following cycle can be included
for i := len(b); i < len(a); i++ {
    a[i] = nil // or the zero value of T
}
a = a[len(b)]
```

Batching with minimal allocation / Расщепление слайса на пакеты данных

фиксированной длины

```
actions := []int{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
batchSize := 3
batches := make([][]int, 0, (len(actions) + batchSize - 1) / batchSize)

for batchSize < len(actions) {
    actions, batches = actions[batchSize:], append(batches, actions[0:batchSize:batch
Size])
}
batches = append(batches, actions)
// batches: [[0 1 2] [3 4 5] [6 7 8] [9]]
```

Move to front, or append if not present, in place / Поиск первого вхождения заданного элемента и перемещение его вперёд (без перераспределения памяти). Если такого элемента не было, то добавляем его в конец слайса.

```
// moveToFront moves needle to the front of haystack, in place if possible.
func moveToFront(needle string, haystack []T) []T {
    if len(haystack) == 0 || haystack[0] == needle {
        return haystack
    }
    var prev T
    for i, elem := range haystack {
        switch {
        case i == 0:
            haystack[0] = needle
            prev = elem
        case elem == needle:
            haystack[i] = prev
            return haystack
        default:
            haystack[i] = prev
            prev = elem
        }
    }
```

```
    return append(haystack, prev)
}
```